CONSTRAINT DATALOG IN TRUST MANAGEMENT

by

Scot Anderson

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Peter Revesz

Lincoln, Nebraska

December 2003

Constraint Datalog in Trust Management

Scot Anderson, M.S.
University of Nebraska, 2003

Advisor: Peter Revesz

Constraint Datalog holds an increasing role in *Trust Management*. We discuss several *Trust Management* systems and give a description of the environment and requirements for *Trust Management*. *Constraint Datalog* using addition constraints and approximation theory provides an expressive semantic with which to describe security policies for credentials, delegations and authorizations. Approximation theory allows halting in *Constraint Datalog* over addition constraints. We use the decision problem of Diophantine equations to show that *Constraint Datalog* over addition constraints is complete. Combining these two concepts provides an approximately complete, safe language. The problem of constant additions to closed languages provides reasons for using an approximately complete, safe language for *Trust Management*. Semantics for the Role-based Trust Management framework (RT) are given in *Constraint Datalog* over addition constraints including an alternate form of a threshold policy.

Acknowledgments

I would like to thank my advisor Dr. Peter Revesz, for his guidance and encouragement during the work of this thesis. I would also like to thank Professor Berthe Choueiry and Professor Byrav Ramamurthy for taking the time to serve on my defense committee.

A special thanks to the faculty at Union College's Division of Business and Computer Science for supporting me throughout my education. And finally I would like to thank my parents, Robert and Charel Anderson, for their support and my wife Patricia Anderson for her encouragement and help in editing my thesis.

Contents

Chapter 1: Introduction

Trust Management is an approach to access control in decentralized, distributed systems with access control decisions based on policy statements made by multiple principals [13]. Decentralized means that an organization involved in *Trust Management* may have several geographical locations where *Trust Management* will be required. The fact that they are distributed means that *Trust Management* may span more than one organization. If there is more than one organization, there will be multiple principals providing authentication and authorization.

Much of the research in the area of *Trust Management* centers around policy expressiveness. As the different frameworks evolved, different security policy situations were suggested that were not expressible in the current languages. This lead to changes or additions to the languages which were in turn updated in a cycle of more research. One may argue that a complete language is not needed as security is certainly a finite discipline with well-known and studied domains. However, computer science is a fast changing field, and what may be adequate today will be obsolete tomorrow.

A Turing-complete language solves the problem of constantly adding capabilities to existing languages but introduces a seemingly insurmountable problem. Namely, in a complete language it is possible to specify a problem for which no answer can be obtained. Since security applications must halt with meaningful results, by using a Turing-complete language we overcome one issue only to face another. Of course that

depends on what the phrase "meaningful results" means. A formal definition of what that means is given in Chapter 3.

Security languages and problems associated with them have been the driving force for the ideas presented in this thesis. We propose that a policy language specification is a variation of the problem expressed by Hilbert in his 10th problem. Namely that a policy language expresses a broad subclass of problems, which by Davis, Putnam, Robinson and Matiyasevich's work [15] is *Turing-complete* hence not decidable. We recognize that halting is absolutely necessary in security applications and propose the use of Datalog with addition constraints to provide proof-based semantics that are expressive (*Turing-complete*) and by using approximation theory also ensure halting. This provides a sound basis on which security and *Trust Management* may be built while allowing extensible semantics that are both approximately complete and safe. The concept of an approximately complete, safe language is very important and it is introduced in Chapter 4.

Approximation theory plays an important part by allowing us to bound the results of any query with only addition constraints. This is vital to ensure halting and we give the theory and an example as part of the background on approximation.

The rest of this thesis is presented as follows. Chapter 2 presents some background information and history of *Trust Management*. In Chapter 3 we discuss the theory of approximation and give an example. In Chapter 4 we show that *Constraint Datalog* over addition constraints is *Turing-complete* and define the concept of an approximately complete, safe language. Chapter 5 examines the Role-based Trust Management (RT) family of languages and gives a translation to Datalog with constraints. In addition in

Chapter 6 we analyze the complexity differences between RT statements and rules in Datalog with constraints. This is followed in Chapter 7 by conclusions and future work.

Chapter 2: Trust Management

Several different language and framework designs have been proposed to solve the *Trust Management* problem. Section 2.1 provides a background look at these languages in the approximate order in which they were proposed. Section 2.2 looks at the environment and its requirements. The world of computers and security is dynamic. It would be the depths of naivety to assume that a system will not change enough to allow new security situations not previously considered. It is important to have an extensible framework. A *Turing-complete* language guarantees extensibility for future policy problems. Section 2.3 introduces the problem of using a *Turing-complete* language as a semantic basis for a security framework. Figure 2.1 gives an graphic overview of the history of Trust management.

2.1 Background of Trust Management

Certificates, reviewed in 2.1.1, form the basis for several *Trust Management* languages. Research at AT&T lead to two *Trust Management* applications, *PolicyMaker*, reviewed in 2.1.2, and *KeyNote*, reviewed in 2.1.3.

2.1.1 Certificates

The concepts of the public key/private key infrastructure are the basis for certificates. Diffie and Hellman created public key cryptography in 1976 [4] and the RSA algorithm was invented by Rivest, Shamir and Adleman in 1977 [20]. Certificates became widely used in the form of X.509 certificates for secure transactions across HTTP on the

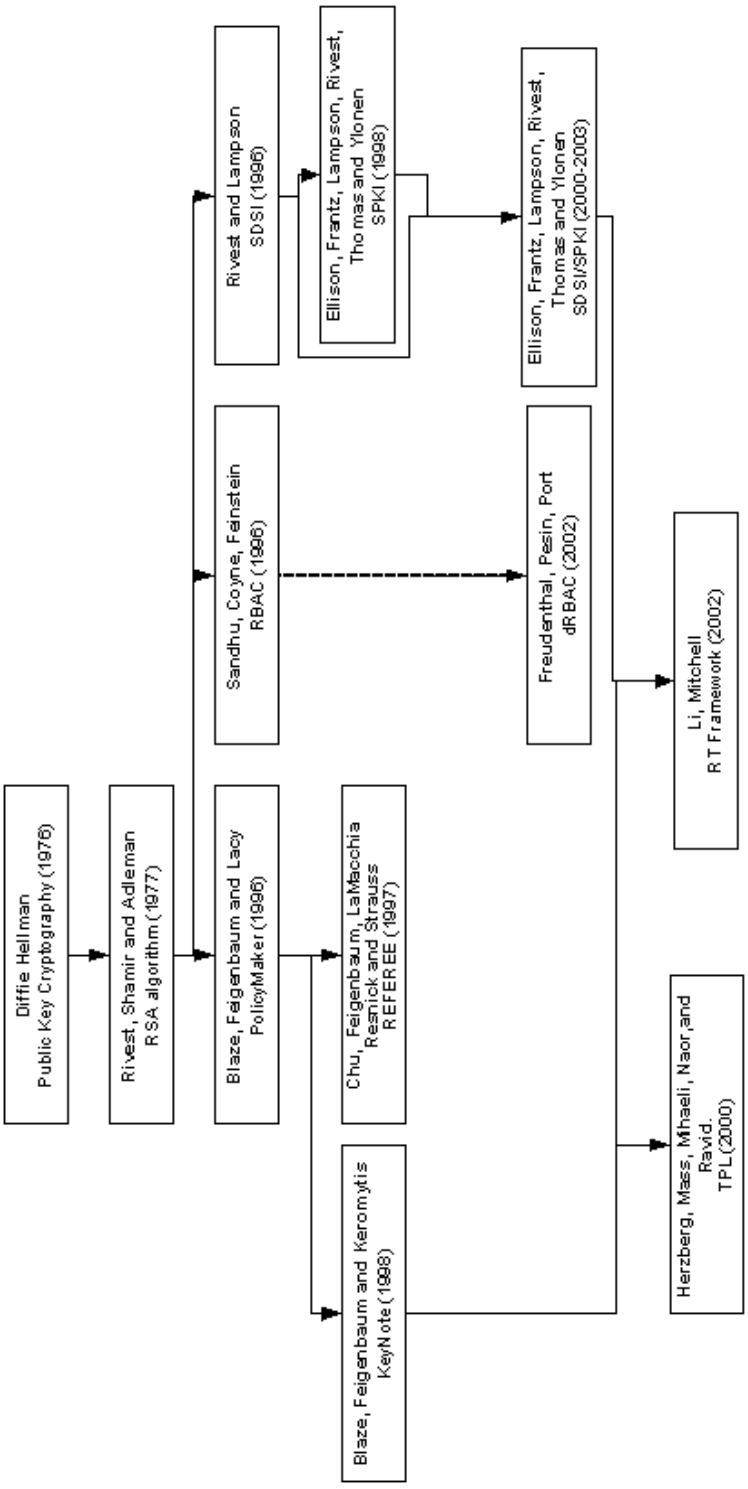

Figure 2.1: A brief history of Trust Management

Internet. PGP certificates were also used for authentication of E-mail transactions. Although X.509 certificates are hierarchical in nature and as such are not suited for *Trust Management*, we will include a brief discussion here as an example of how certificates are employed in the context of *Trust Management*. X.509 certificates contain several basic elements as shown in Table 2.2.

Version Number
Serial number
Certificate holder's distinguished name
Validity period
Certificate holder's public key
Algorithm to use for communication
Certificate issuer's name
Digital signature of the certificate authority

Figure 2.2: X.509 Fields

The *version number* indicates what version of the X.509 standard this certificate complies with. The *serial number* is a unique number provided by the certificate authority (CA) used to identify the certificate and to check revocation lists. The *certificate holders distinguished name* is meant to be unique across the Internet and is usually associated with a domain name. The *validity period* indicates when this certificate expires. The *certificate issuer's name* is the name of the CA that signed the certificate. These elements are simple and have simple tasks.

The *certificate holder's public key* is part of an asymmetric public/private key pair. They are asymmetric because information encrypted by a private key can only be decrypted by the public key and vice versa. This allows secure communication from anyone using the public key to the holder of the private key. *The algorithm to use in communication* is the method the holder of this certificate is expecting when the public key is used to encrypt communications. This is the basis for secure communication.

The *digital signature* guarantees that the CA has issued this certificate and that the information contained in the certificate is valid. This concept is called non-repudiation, which proves that the certificate in its current verified form came from the CA and can be trusted. The digital signature is actually a one-way hash code of the certificate. Thus you can check it against a calculated hash code of the certificate. If the two are the same, you know that the certificate has not been tampered with and that if you use the public key contained in the certificate to encrypt communications, then only the holder of the private key can decrypt your communication.

2.1.2 PolicyMaker

PolicyMaker was the first *Trust Management system* [8]. It has *policies* and *credentials* and allows any safe *assertion language* to be used. (An *assertion language* is a language that allows the specification of a set of assertions.) An *assertion* has the form

$$source \text{ ASSERTS } AuthorityStruct \text{ WHERE } Filter$$

Source is either a key or *POLICY* for local policies. $Authoritystruct$ is the key of the target of the assertion. *Filters* are just interpreted programs that determine if a string is

accepted or rejected. The meaning of the assertion statement is that the source asserts that the individuals identified in the $AuthorityStruct$ have rights granted by the filter.

An application gives *PolicyMaker* a set of requested actions, policies and credentials and *PolicyMaker* attempts to prove requested actions comply with the policies. The major weakness of *PolicyMaker* is that much of the work in evaluating requests is left up to the calling application. The filter language is not specified and the authentication of certificates is not done by *PolicyMaker*.

2.1.3 KeyNote

KeyNote [1] overcomes some of the weaknesses of *PolicyMaker* and includes two additional design goals: *standardization* and *ease of integration*. In *KeyNote* more is done by the *Trust Management* engine than in the calling application. Signature verification is done by *KeyNote*, and a specific assertion language is used. An *Action Environment* in the form of $attribute = value$ pairs provide *KeyNote* with the relevant information about an application's security requirements. Since *KeyNote* was built on *PolicyMaker* technology, it retains the same design principles of assertions and queries.

2.1.4 REFEREE

REFEREE [2] which stands for *Rule-controlled Environment For Evaluation of Rules and Everything Else* was also based on *PolicyMaker*. It was designed to work with *PICS* a W3C standard for rating web pages. Like *PolicyMaker* it accepted information including *PICS* certificates and returned a value indicating whether the information and policies allow or deny access. Unlike *PolicyMaker*, *REFEREE* provides a third

alternative called *unknown* stating that it could not determine whether access is allowed. *REFEREE* appears to be unique in this respect. In a way it recognizes that not everything is true or false, but some things are unknown.

2.1.5 TPL

Trust Policy Language (TPL) [11] was designed at IBM as a role-based solution to e-commerce *Trust Management* problems. *TPL* uses X.509 v3 certificates, the *TPL* language, and Java. The *TPL* language component is defined in XML. The important difference between *TPL* and *PolicyMaker* is that it allows *negative* rules preventing access. We note that *DTPL (Definite Trust Policy Language)* [9] does not allow negative rules. The primitive structure for *TPL* is a group. The syntax in XML is different enough from *PolicyMaker* to warrant an example. Consider the policy in Figure 2.3 which is from [11].

The first group defined is the originating retailer *self*. The second group, *partner* includes anyone with a partner certificate signed by *self*. The third group, *departments* includes partner certificates signed by the partner group. The last group defines the *customers* group to be anyone with an employee certificate signed by the departments group. The function tag defines constraints using AND, OR, NOT, GT(>), and LT(<). For example, the <FUNCTION> tag is a constraint where the *rank* field on a customer certificate is greater than 3.

```
<POLICY>
   <GROUP NAME="self" />
   <GROUP NAME="partners">
      <RULE>
         <INCLUSION ID="partner" TYPE="partner" FROM="self" />
      </RULE>
   </GROUP>
   <GROUP NAME="departments">
      <RULE>
         <INCLUSION ID="partner" TYPE="partner" FROM="partners" />
      </RULE>
   </GROUP>
   <GROUP NAME="customers">
      <RULE>
      <INCLUSION ID="customer" TYPE="employee" FROM="departments" />
         <FUNCTION>
            <GT>
               <FIELD ID="customer" NAME="rank" />
               <CONST>3</CONST>
            </GT>
         </FUNCTION>
      </RULE>
   </GROUP>
</POLICY>
```

Figure 2.3: TPL Policy

2.1.6 dRBAC

dRBAC (Distributed Role-based Access Control) [6] started as RBAC [21] in 1996 and has expanded to be a fully distributed access control system. dRBAC concentrates on highly dynamic coalition environments where the ability to adapt quickly, scale and find credentials is needed. dRBAC distinguishes itself in the following three ways.

1. Third-party delegation of roles from outside a domain's name space rely on an explicit delegation of assignment.

2. Modulation of transferred permissions takes place using scalar value attributes associated with roles.

3. Continuous monitoring of trust managements over long-lived interactions is implemented.

In item (2) above, scalar value attributes are similar to parameterized roles in RT and allow dRBAC to include ancillary information about the authorization such as bandwidth restrictions.

2.1.7 SPKI/SDSI

SPKI/SDSI [5] is defined in RFC2693. It uses a localized naming scheme and uses public keys to identify principals and local identifiers. These name-definition certificates came from *SDSI*. Authorization certificates came from *SPKI*. Globally unique names require globally unique identifiers for the organizations. That is easily accomplished by prepending a URI to the beginning of local names.

Although *SPKI/SDSI* shares many goals with *Trust Management* systems, Li points out in [12] that much of the processing of certificates to authorize requests is left to application developers. Hence *SPKI/SDSI* falls short of the *Trust Management* ideal.

2.1.8 Role-based Trust Management (RT)

RT [14] is a complete *Trust Management* system that uses certificates for naming and includes authorization on the certificates. Only certificates and requests are required for RT to accept or reject a request. In Chapter 5 we discuss *RT* in more detail.

2.2 Requirements for Trust Management

For the purpose of fully understanding the requirements of an open, decentralized, distributed environment, it is necessary to define what is meant by such an environment, and then incorporate these characteristics into the expressive capabilities of the *Trust Management* system.

2.2.1 Description of the Environment

The first qualifier *open* refers to the ability for anyone to be able to make a request for access or service to a supplier of the same. For example, anyone who has a valid user name with an Internet provider can type in the URL http://etrade.com and be connected to the site. In this example, fewer services are available to non-members. In order to make use of the main brokerage and banking services provided by e-trade, it is necessary to provide your e-trade user ID and password. The fact that non-members can gain access to the login page of the e-trade service makes it *open*.

The second qualifier, *decentralized*, refers to the various resources being located in physically separate locations. An example of this would be the Yahoo site, which is physically located on many different servers across the world.

The third qualifier *distributed* refers to the situation where independent organizations enter into coalitions to share resources. While sharing resources, each autonomous member retains ultimate authority over the resources it controlled prior to entering the coalition. A familiar example of this would be the ubiquitous *ATM* unit. These are decentralized in that one bank owns many *ATMs* located in different areas. They are also distributed in that authorization is granted to account holders from other banks as well as to account holders from the bank that owns the *ATM*.

Combining all three of these descriptors results in multiple resources that are located in multiple locations, independently controlled by multiple services and open for requests by the public. In these situations, the owners and requester may be unknown to each other. The *Trust Management* System needs to protect both the service providers and the user of the service. The *Trust Management* system must cover a wide variety of applications without requiring a custom package for each. The ability to incorporate a large range of security situations within a single *Trust Management* system is referred to as the *expressiveness* of the *Trust Management* system.

We will address the issue of expressiveness by first describing the expanding needs of the *open*, *decentralized*, *distributed* environment and showing how existing *Trust Management* systems begin to falter in function.

2.2.2 Requirements of the Environment

X.509 certificates fill several requirements of a *Trust Management* system. Among these are secure communication between organizations and users and provision for authentication of both the requester and the service provider. Although X.509 certificates provide a distributed users database, it does not provide distributed authentication of that database because the ultimate authority is the root certificate authority. X.509 certificates can not authorize access unless all you are interested in is authenticating the user.

Because most *Trust Management* systems use certificates in some form or another, the emphasis on distributed authorization can be seen in the following *Trust Management* requirements.

1. An organization should have localized control of its resources. This entails granting access and authority to parties known to the organization, e.g., owners, employees, service personnel and contractors.

2. An organization should be able to grant either full or partial access and delegate authority to other members of a coalition to which it belongs.

3. An organization should be able to have delegation of either full or partial access and authority to the resources belonging to other members of a coalition to which it belongs.

4. The *Trust Management* system should be able to operate regardless of the security domains in place among members of a coalition such as hierarchical or group based domains. Domains also pertain to the way security is administrated at any location.

For example, whether a user is provided with a USB key or must use a password system, the *Trust Management* system should be able to operate as a back-end system to the application or system needing to verify authorization.

5. An organization should be able to enter into multiple coalitions independent of each other but as interlaced as desired.

6. An organization should be able to delegate, not only the authorization but also the right to delegate authorization. We call this *nested trust delegation*.

Requirements (4) and (6) are open ended in a way that can cause expressiveness problems for current languages. One could argue that policies will only cover a finite definable area, and that languages can be extended on a case by case scenario, which has been the approach in *Trust Management* to this point. Certainly in the majority of systems we can cover the cases where security needs to be supplied. To avoid constantly changing a language or adding to it, we could use a Turing-complete language.

2.3 Problems with a Complete Language

A *Turing-complete* language suffers from the halting problem [16], and in security it is absolutely necessary to halt in a reasonable amount of time. If we approximate the results of a query to ensure halting, then it must not allow access beyond what is intended. *Datalog*, which is based on *First-order Predicate Logic* easily accomplishes halting because it operates under a closed world assumption. Essentially a proof is constructed from the data D using a query Q to satisfy a policy R. If $\vdash_{Q,D} R$, then we can answer

yes to the access request. But Datalog is not complete. *Constraint Datalog* with addition constraints also has a proof based semantic, and it is a complete language as is shown in Chapter 4. Using approximation we can overcome the difficulties of halting in a complete language.

Chapter 3: Approximation

Approximation theory allows a query to be approximated over a finite domain. In Sections 3.1 and 3.2 we give the theory of approximation. In Section 3.3 we present an example based on Sierpinsky's Carpet.

3.1 Approximation Theory

Unlike relational databases where each query operates under a closed world assumption, *Constraint Datalog* allows recursion that may not terminate [17]. A good example of a query that may not terminate is the following pair of rules that define the difference relation D.

$$\begin{array}{lll} D(x,y,z) :- & x-y \leq 0, \ -x+y \leq 0 & \\ & z \leq 0, \ -z \leq 0 & \\ D(x,y,z) :- & D(x',y,z'), & \\ & x-x' \leq 1, \ -x+x' \leq -1, & \\ & z-z' \leq 1, \ -z+z' \leq -1, & \end{array} \tag{3.1}$$

Approximation allows us to halt with an approximation by bounding the result of addition constraints. This limits us to addition constraints, but as we will see later this does not reduce the expressiveness of *Constraint Datalog*. We define the following two modifications to addition constraints taken from [18].

Definition 3.1 *Modification 1: We change in the constraint tuple the value of any bound b to be the $min(b,u)$. Given a query Q, an input database D and an upper bound u, the query using this modification will be denoted $Q(D)_u$.*

Definition 3.2 *Modification 2: We delete from each constraint tuple any constraint with a bound that is greater than u. Given a query Q, an input database D and an upper bound u, the query using this modification will be denoted $Q(D)^u$.*

Using these two modifications we can express the important result that we can bound the least fixed point of a *Constraint Datalog* query.

Theorem 3.3 *For any Datalog query Q, input database D, and constant $u > 0$, the following is true.*

$$Q\left(D\right)_u \subseteq lfp(Q\left(D\right)) \subseteq Q\left(D\right)^u \tag{3.2}$$

Further, $Q\left(D\right)_u$ and $Q\left(D\right)^u$ can be computed in finite time.

Proof The Datalog bottom-up evaluation can be done followed by either modification 1, 2, or no modification if the constant bound b is less than u. If the rule is modified by the first modification, the modified tuple implies the original one, or it implies an unsatisfiable statement. If the result of the modification is false, then by definition it is not part of $Q\left(D\right)_u$. And so, by the definition of the least fixed point, all facts in $Q\left(D\right)_u \subseteq lfp\left(Q\left(D\right)\right)$. If a rule is modified by the second modification, the result will be a new fact of the same form except that it does not contain upper bounds. This allows the addition of facts which need not be in $lfp\left(Q\left(D\right)\right)$. However these new facts will produce only facts that are subsumed by facts in $Q\left(D\right)^u$. Thus for any number of rule applications, the facts in $Q\left(D\right)^u$ are implied by previous facts obtained as part of the $lfp\left(Q\left(D\right)\right)$. Thus $lfp\left(Q\left(D\right)\right) \subseteq Q\left(D\right)^u$.

Since there is a finite number of different left-hand side atomic addition constraints and since each rule application will change the bound a constant amount closer to the lower bound, $Q(D)_u$ and $Q(D)^u$ will both be evaluated in finite time. ■

3.2 Defining Meaningful Results for Trust Management

Meaningful results for a *Trust Management* system is defined in terms of Theorem 3.3 using Modification 1. Namely the results of the approximation must contain only true statements.

Definition 3.4 *A result from a Trust Management system is considered meaningful if it does not grant more privileges than have been granted by the principal responsible for the resource being requested.*

3.3 Fractal Example

Applying approximation to Fractals [7] is a natural problem. Fractals by nature show ever more detail as we zoom in. In fact we can zoom in an infinite amount in a mathematical model. When fractals are generated on screens or in print, they are approximations of the mathematical definitions used to generate them. The most interesting property about fractals is their repetition on a smaller and smaller scale.

Sierpinski's carpet [7] is a special fractal that can be approximated by starting with the smallest scale and building the fractal up to the limit of approximation. Consider the *Constraint Datalog* program written for MLPQ [19] in Table 3.1.

```
begin%Boxes%
Choice(m,n) :- m=0, n=0.
Choice(m,n) :- m=0, n=1.
Choice(m,n) :- m=0, n=2.
Choice(m,n) :- m=1, n=0.
Choice(m,n) :- m=1, n=2.
Choice(m,n) :- m=2, n=0.
Choice(m,n) :- m=2, n=1.
Choice(m,n) :- m=2, n=2.
D(x,y,z) :- x-y=0, z=0.
D(x,y,z) :- D(x1,y,z1), x-x1=1, z-z1=1.
M(x,y,z) :- x=0, y=0, z=0.
M(x,y,z) :- M(x1,y,z1), D(z,z1,y), x-x1=1.
M(x,y,z) :- M(x,y1,z1), D(z,z1,x), y-y1=1.
/* Creates the first set of base lines length=1,3,9...*/
BaseLine(x1,x2):- x1=0, x2=1.
BaseLine(a1,d3):- BaseLine(a1,d1), l=3, M(l,d1,d3).
Box(x,y,s) :- BaseLine(x,b), y-x=0, b-s=0.
Box(x,y,s) :- Box(a,b,l), k=3, M(k,s,l), Choice(m,n),
              M(s,m,z1), D(x,a,z1),
              M(s,n,z2), D(y,b,z2).
XBox(a1,c1,s) :- Box(a,c,l), k=3, M(k,s,l), D(a1,a,s),
                 D(c1,c,s).
fractal1(id,x,y) :- id=1, Box(a,b,l), x-a>=0, D(x1,l,a),
                    x-x1<=0, y-b>=0, D(y1,l,b), y-y1<=0.
fractal2(id,x,y) :- id=2, XBox(a,b,l), x-a>=0, D(x1,l,a),
                    x-x1<=0, y-b>=0, D(y1,l,b), y-y1<=0.
end%Boxes%
```

Table 3.1: MLPQ Fractal Program

The Difference relation equation (3.1) is really the basis for approximation. We do not allow the calculation of differences for some maximum bound in z. In Sierpinski's carpet the Multiplication relation M(x,y,z) is based on D(x,y,z). We use approximation and M(x,y,z) to find the width and height of the biggest box in the BaseLine(x1,x2) relation. Here we just find the largest power of 3 that approximation allows. Based on these lengths we create the first set of boxes and use a recursive rule to create the bottom, left point and width and height of each box. XBox(a1,c1,s) creates the interior boxes of the fractal. Running the above program on MLPQ using an approximation value of 27, we get Sierpinski's carpet as shown in Figure 3.1.

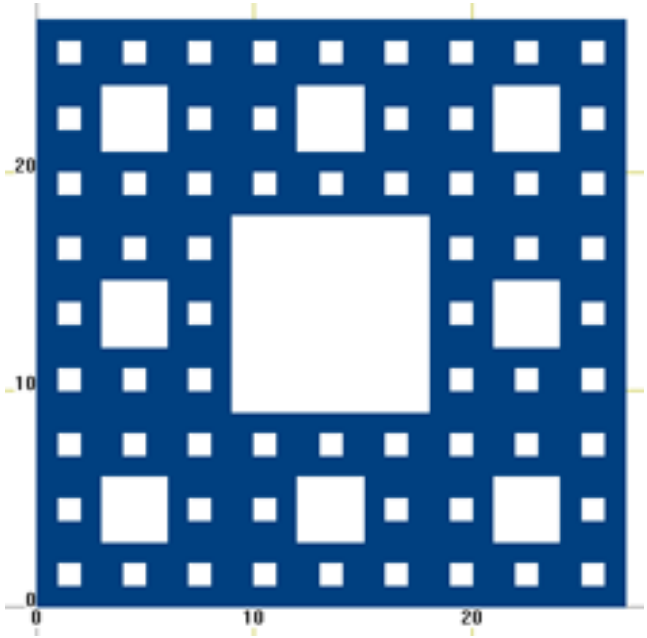

Figure 3.1: Sierpinski's Carpet

Chapter 4: Turing Completeness of Constraint Datalog

Turing's definition of an algorithm gives a basis on which other languages can be judged. A language is *Turing-complete* if it can express everything that a Turing machine can express. That is important because if a language is *Turing-complete*, then it can express everything that a computer is capable of expressing. To show that a language is *Turing-complete* it is necessary to define a reduction from a Turing Machine to the language in question. Alternately, a *Turing-complete* language can be reduced to the language in question. This second method is used to show that a *Constraint Datalog* is *Turing-complete* using Diophantine equations.

4.1 Turing Machines

A Turing machine [10] is an abstract computer. It has memory in the form of a tape that starts at the machine and has an infinite supply. That is unique because a Turing machine will never run out of memory. The tape is divided into cells that contain symbols from a finite set $A = \{\alpha_1, \alpha_2, ..., \alpha_w\}$. Different Turing machines may have different alphabets, but all Turing Machines will have a special symbol $*$ to mark the start of the tape. Turing Machines also contain a symbol Λ for an empty cell. The Turing machine has a read/write head that reads and writes to a cell on the tape. In addition, the head can move left or right across the tape.

At any given time, the machine is in one of finitely many states that are denoted by $q_1, q_2, ..., q_v$. The action of the machine is totally determined by the current state and the symbol scanned by the head. In a single step, the machine can change the symbol in the current cell or move one cell to the right or left and change state. Given a state q_i and a symbol read from the tape α_j the next state is defined as:

$$q_i \alpha_j \Rightarrow \alpha_{A(i,j)} D\left(i,j\right) q_{Q(i,j)} \tag{4.1}$$

where

- $\alpha_{A(i,j)}$ is the symbol to be written
- $D\left(i,j\right)$ represents the motion of the head such that $D\left(i,j\right) \in \{L, R, S\}$ for left, right and stay.
- $q_{Q(i,j)}$ is the new state.

4.2 Diophantine Equations are Turing-complete

Here we present the sequence of theorems and thoughts that lead Matiyasevich [15] to answer that Hilbert's 10th problem, that is, the decision problem for Diophantine equations, is undecidable. The consequence of solving that major question has many other applications and consequences. Specifically we see that Diophantine equations are used to express listable sets (see Definition 4.3). That is, listable is a sufficient condition for a set to be Diophantine. This remarkable result proves that Diophantine sets are expressively equivalent to a Turing machine. We then show that *Constraint Datalog*

over addition constraints is *Turing-complete*. We conclude that *Constraint Datalog* with approximation over addition constraints is safe.

A Diophantine equation is an equation of the form

$$D\left(x_1, ..., x_m\right) = 0, \tag{4.2}$$

where D is a polynomial with only integer coefficients and integer variables.

It is important to note that this decision problem is equivalent to allowing only natural numbers. Consider an equation of the form:

$$D\left(p_1 - q_1, ..., p_m - q_m\right) = 0 \tag{4.3}$$

where $p_i, q_i \in \mathbb{Z}^+$. It is easy to see that if we can find a solution to equation (4.3) then we have found a solution to equation (4.2) by setting $x_i = p_i - q_i$ for $1 \leq i \leq m$. It is equally true that if we have an equation of form in equation (4.2), then there exists an equivalent equation with only natural number solutions. Therefore the decision problem for Diophantine equations with integers is equivalent to the decision problem for Diophantine equations with natural numbers.

Families of Diophantine equations are denoted by the existence of parameters $a_1, ..., a_n$ and have the form:

$$D\left(a_1, ..., a_n, x_1, ..., x_m\right) = 0. \tag{4.4}$$

We consider the set $\mathcal{M}$ of all n-tuples $\langle a_1, ..., a_n \rangle$ for which our parametric equation has a solution, that is

$$\langle a_1, ...a_n \rangle \in \mathcal{M} \Longleftrightarrow \exists x_1 ... x_m \left\{D\left(a_1, ..., a_n, x_1, ..., x_m\right) = 0\right\} \tag{4.5}$$

Sets having such a representation are called **Diophantine**.

In studying Hilbert's tenth problem it was natural to take an in-depth look at the properties for these sets. What kind of sets can we find and how can we list them? One important property is that Diophantine sets are listable.

Definition 4.1 [15] *A set M is listable or effectively enumerable, if there exists an algorithm which would print in some order, possibly with repetitions, all the elements of the set M.*

Martin Davis conjectured that Diophantine sets and listable sets coincide.

Conjecture 4.2 [15] *A set S is Diophantine if and only if S is listable.*

This implies that any set that is listable can be obtained from a single polynomial. Davis's conjecture also implied the existence of a Universal parameterized Diophantine equation that for some parameters would produce any specific listable set. This is the result that is most interesting to us. If Conjecture 4.2 is true, then Diophantine equations are expressively equivalent to Turing machines, and there exists a universal parameterized Diophantine equation that is expressively equivalent to a universal Turing machine.

Martin Davis made the first strides toward proving his conjecture. He showed the following theorem.

Theorem 4.3 *Every listable set M has a representation of the form*

$$\langle a_1, ..., a_n \rangle \in \mathcal{M} \Leftrightarrow \exists z \forall y \leq z \exists x_1 ... x_m \{D(a_1, ..., a_n, x_1, ..., x_m, y, z) = 0\} \qquad (4.6)$$

This became known as the Davis normal form. All that was left to do was remove the last

universal qualifier. Davis, Putnam and Robinson obtained an exponential Diophantine representation for every listable set in the following theorem.

Theorem 4.4 *For every listable set M of n-tuples of non-negative integers there is a representation of the following form*

$$\langle a_1, ..., a_n \rangle \in \mathcal{M} \Leftrightarrow \exists x_1 ... x_m E_L (a_1, ..., a_n, x_1, ..., x_m) = E_R (a_1, ..., a_n, x_1, ...x_m)$$

where E_L and E_R are exponential polynomials.

Thus we have a purely existential representation for all listable sets. But it is not Diophantine, but rather it is exponentially Diophantine. One of the more interesting implications of Theorem 4.4 is that you can fix the number of unknowns for any arbitrary exponential Diophantine equation to be as low as 3 unknowns.

Based on this work Julia Robinson found a condition sufficient for the existence of a Diophantine representation.

There is a polynomial $A (a, b, c, z_1, ..., z_m)$ such that

$$a^b = c - \exists z_1 ... z_w \{A (a, b, c, z_1, ..., z_m) = 0\} \tag{4.7}$$

provided that there is an equation

$$J (u, v, y_1, ..., y_w) = 0 \tag{4.8}$$

such that in every solution we have $u < v^v$ and for every k there is a solution such that $u > v^k$.

Now to prove Conjecture 4.2 it only remained to show that there exists a single relation of exponential growth defined by a Diophantine equation. Yuri Matiyasevich who finally proved this using the well known Fibonacci numbers. The relation between

u and v in the Julia Robinson relation is

$$v = F_{2u} \tag{4.9}$$

where $F_1, F_2, ...$ is the well known Fibonacci sequence:

$$0, 1, 1, 2, 3, 5, 8, ... \tag{4.10}$$

This proved Conjecture.4.2 resulting in the in the following theorem.

Theorem 4.5 [15] *Every listable set M of n-tuples of non-negative integers has a Diophantine representation, that is*

$$\langle a_1, ..., a_n \rangle \in \mathcal{M} \Leftrightarrow \exists x_1 ... x_m \left\{ D\left(a_1, ..., a_n, x_1, ..., x_m\right) = 0 \right\} \tag{4.11}$$

for some polynomial with integer coefficients.

The definition of listable sets in Definition 4.3 and the results from Theorem 4.5 allows us to conclude that Diophantine equations are *Turing-complete*.

4.3 Expressing Diophantine Equations in Constraint Datalog

Constraint Datalog over addition constraints with approximation provides the ability to express Diophantine equations and to approximate the results. The interesting result here is that one can approximate results to the same level that a computer is capable of approximating results. Again here the limit is storage space. First we define a difference relation and a multiplication relation in Table 4.1

Given any Diophantine equation of the form $D\left(a_1, ..., a_n, x_1, ...x_m\right)$ with k terms we collect the expressions in each term into a single variable T_i for $1 \leq i \leq k$ using the

```
D(x,y,z) :- x-y=0, z=0.
D(x,y,z) :- D(x1,y,z1), x-x1=1, z-z1=1.
M(x,y,z) :- x=0, y=0, z=0.
M(x,y,z) :- M(x1,y,z1), D(z,z1,y), x-x1=1.
M(x,y,z) :- M(x,y1,z1), D(z,z1,x), y-y1=1.
```

Table 4.1: Difference and Multiplicaton Relations

multiplication relation. If we expand exponents to multiplication in each term, we write the collection as follows.

Given a term with c constants and variables v_l after the expansion we can collect the constants and variables by writing a series of multiplication statements in Datalog. $T_{i,1} = v_1 v_2, T_{i,2} = T_{i,1} v_3, ..., T_{i,c-1} = T_{i,c-2} v_c$. In general we write

$$T_{i,l} = T_{i,l-1} v_{l+1} \text{ for } 2 \leq l \leq c-1 \tag{4.12}$$

Next we collect successive term variables T_i into expression variables E_j for $1 \leq j \leq k-1$. We set $E_1 = T_1 - (\pm T_2)$ for the first expression and $E_j = E_{j-1} - (\pm T_{j+1})$. In Datalog we write this

$$D\left(Ej, E_{j-1}, \pm T_{j+1}\right) \text{ for } 2 \leq j \leq k-1 \tag{4.13}$$

Example 4.6 *Given the equation* $A\left(x,y,z\right) = 3xy^2 + 2z$*, solve* $A\left(x,y,z\right) = 0$.

Using the Difference and Multiplication examples above we write

$$\begin{aligned} A\left(x,y,z\right) :- \quad & M\left(3,x,t11\right) \\ & M\left(t11,y,t12\right) \\ & M(t12,y,t13) \\ & M\left(2,z,t21\right) \\ & M\left(t21,-1,t22\right) \\ & D\left(t13,t22,0\right) \end{aligned}$$

We illustrate the procedure in a parse tree as shown in Figure 4.1. The circles denote either the Multiplication or Difference operations at each step.

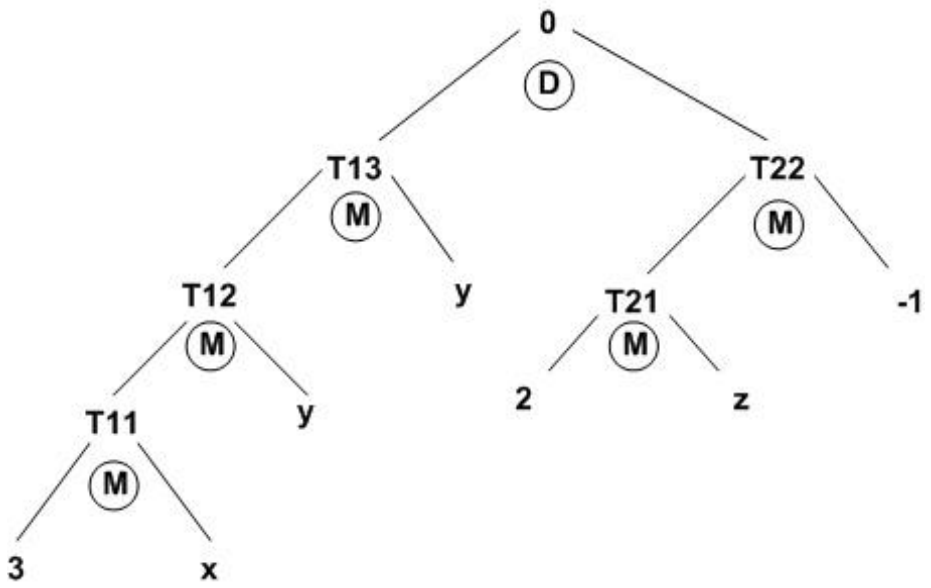

Figure 4.1: Parse Tree for Diophantine Equations

In this manner we can write any Diophantine equation in a finite number of steps using approximation. The reduction from a Diophantine equation to *Constraint Datalog* is linear in the number of multiplication, addition and subtraction operations. Therefore we conclude that *Constraint Datalog* with addition constraints is *Turing-complete*.

Languages that are both complete and safe with approximation have real value in the area of security and *Trust Management*. We propose the following definition.

Definition 4.7 *The Semantics of a language that uses approximation to terminate an evaluation of a program while retaining meaningful results over a definable finite subdomain of the original domain of the program and is expressively Turing complete is an approximately complete, safe language.*

Datalog over addition constraints with approximation is an approximately complete, safe language.

From Theorem 4.5 we know that Datalog with addition constraints can express any Diophantine set and thus is Turing complete. From Theorem 3.3 we know that Datalog over addition constraints with approximation uses approximation to terminate an evaluation of a program while maintaining meaningful results as defined in Definition 3.4. The difference relation equation (3.1) provides the definition of the subdomain of integers. Therefore Datalog over addition constraints with approximation is an approximately complete, safe language.

An approximately complete, safe language allows the language to perform a graceful termination instead of a machine imposed termination due to the lack of memory. To control termination requires either direct user specification or machine/operating system cooperation with the language. This leads us to define one last definition.

Definition 4.8 *An approximately complete, safe system exists if there exists some set of parameters provided by the hardware, operating system or other source to an approximately complete, safe language that allows the language to determine an appropriate approximation threshold.*

As a thought example we could ask an approximately complete, safe system to give us a representation of π. The system would then determine, based on the hardware, operating system and parameters such as time constraints, which are set by the operator, how many digits it could calculate.

Definitions 4.7 and 4.8 will form the basis for future research.

Chapter 5: Translating a Trust Management Framework into Constraint Datalog

5.1 The RT Family of Trust Management

RT is a role-based trust-management family of languages part of which is based on Datalog with constraints. The RT language includes principals and roles. A principal may be a uniquely identified individual or process and may issue policy statements and make requests. At any time RT is capable of determining which principal made a particular statement or request. Roles are defined by a $PrincipalName$ followed by a $RoleName$. Roles act as a layer between principals and permissions and are similar to groups. Given $K_A.R$, we say that principal K_A defines the role R. Each entity has the authority to define a role and the members of that role. A role is defined by one or more statements with the effect being the union of the groups defined. Before we give the syntax and semantics of RT lets look at the types of delegation possible.

- RT_0 supports localized authorities for roles, role hierarchies, delegation of authority over roles, attribute based delegation of authority, and role intersections.
- RT_1 adds parameterized roles to RT_0
- RT_2 adds logical objects to RT_1

- RT^T provides *manifold roles* and *role-product* operators, which can express *threshold* and *separation-of-duty* policies.

- RT^D provides delegation of role activations, which can express selective use of capacities and delegation of these capacities.

5.1.1 RT_0 Syntax

Four different types of statements are necessary to define all the role types available in RT_0.

Simple Member: This defines the principal K_D to be a member of the role $K_A.R$

$$K_A.R \leftarrow K_D \tag{5.1}$$

Simple Containment: This defines the role $K_A.R$ to contain every principal that is a member of the role $K_B.R_1$.

$$K_A.R \leftarrow K_B.R_1 \tag{5.2}$$

Linking Containment: This defines $K_A.R$ to contain all members of $K_B.R_2$ in which K_B is a member of $K_A.R_1$.

$$K_A.R \leftarrow K_B.R_1.R_2 \tag{5.3}$$

Intersection Containment: This defines $K_A.R$ to contain the intersection of all the roles $K_{B_1}.R_1 ... K_{B_k}.R_k$

$$K_A.R \leftarrow K_{B_1}.R_1 \cap ... \cap K_{B_k}.R_k \tag{5.4}$$

To complete the picture, the two types of statements below specify the syntax for delegation. Delegations can be created by combinations of the previous four statements but were added for clarity.

Simple Delegation: This means that K_A delegates authority over R to K_B. We can optionally impose that K_B can only authorize existing members of $K_C.R_2$. If $K_C.R_2$ does not exist, the imposition does not apply.

$$K_A.R \Leftarrow K_B : K_C.R_2 \tag{5.5}$$

This implies

$$LogK_A.R \leftarrow K_B.R \cap K_C.R_2 \tag{5.6}$$

Linking Delegation: This means that $K_A.R$ delegates authority over R to members of $K_A.R_2$. Optionally the delegation is restricted to members of $K_C.R_2$.

$$K_A.R \Leftarrow K_A.R_1 : K_C.R_2 \tag{5.7}$$

This implies

$$K_A.R \leftarrow K_A.R_1.R \cap K_C.R_2 \tag{5.8}$$

5.1.2 RT_1^C Adding Parameterized Roles

RT_1^C also provides parameterized roles. Parameters are an extension of RT_0. The format is similar to the statements in Section 5.1.1 with the exception that a role takes the form $r\,(p_1, ..., p_n)$, in which r is the role name and p_j can take one of the following three forms.

$$\begin{array}{l} name = c \\ name = ?X\ [\in S] \\ name \in S \end{array}$$

These respectively mean that a name equals to 1) a constant, 2) a variable with optional requirement that it be in a set S, and 3) is in a set S.

The form of the parameters determines their function. Parameterized roles can represent relationships between entities or represent access permissions that take parameters identifying resources and access modes [4]. For instance,

$$Store.Employee(Manager = "Steve") \leftarrow "Charlie"$$

states that Charlie is an employee and his manager is Steve. At some point Steve could give Charlie a pay increase, because he is listed as Charlie's manager. Another example is:

$$Web.perm(host \in descendant(\text{'unl.edu'}), port \in [0..1024]) \leftarrow "Charlie"$$

Which shows how the element operator is used with two different types of constraint domains. The expression means that Charlie is given access to any host in the unl.edu domain on privileged ports. In this example host is in a tree domain, and port is in a range domain.

5.1.3 RT_2 Logical Objects

Logical objects are added in the guise of *o-sets* which perform similar duties to roles except they do not contain principals, they apply to logical objects. This means that we can group logical objects together with principals and permissions. An *o-set* id has a specific type τ associated with it. The body of a *o-set* definition can have a value of base type τ, another *o-set* of type τ, a linked *o-set* similar to role linking or an intersection of k *o-sets*. These rules are as follows:

Value assignment: This means that v is a member of the *o-set* $A.o$. Note that v could be an expression that evaluates to a value of type τ.

$$A.o\,(p_1, ..., p_n) \leftarrow v \tag{5.9}$$

o-set **assignment:** This just adds the contents of one *o-set* to another.

$$A.o\,(p_1, ..., p_n) \leftarrow B.o_1\,(v_1, ..., v_m) \tag{5.10}$$

Linked *o-set*: Here the *o-set* o contains any logical object that is defined by any principal defining o_1 in A's role r_1.

$$A.o\,(p_1, ..., p_n) \leftarrow A.r_1\,(t_1, ..., t_l)\,.o_1\,(v_1, ..., v_m) \tag{5.11}$$

Intersection of *o-sets*: Here we just define the intersection of k *o-sets*

$$A.o\,(p_1, ..., p_n) \leftarrow B_1.o_1\,(v_1, ..., v_{m_1}) \cap ... \cap B_k.o_1\,(v_1, ..., v_{m_k}) \tag{5.12}$$

Consider the example given in [14].

Example 5.1 *Alpha allows the manager of the owner of a file to access that file:*

$$Alpha.read\,(?F) \leftarrow Alpha.manager\,(?E : Alpha.owner\,(?F))$$

Here $?E$ *is assigned the output of* $Alpha.owner\,(?F)$ *for every file* $?F$.

5.1.1 RT^T Manifold Roles

Manifold roles extends the notion of a role to include an entity collection. Since a role is a set of principals or entities, and this is extended to contain not only entities but collections of entities or sets of entities this allows a set of sets. To have a consistent

semantics, Manifold roles are used in the case of roles in RT_i by replacing single entities with a set containing just that one entity. Two new types of *credentials* are introduced in RT^T

Manifold dot products:

$$A.R \leftarrow B_1.R_1 \odot ... \odot B_k.R_k \tag{5.13}$$

This syntax means

$$A.R \supseteq \{s_1 \cup ... \cup s_k | s_i \in members\left(B_i.R_i\right), 1 \leq i \leq k\} \tag{5.14}$$

Manifold cross products:

$$A.R \leftarrow B_1.R_1 \otimes ... \otimes B_k.R_k \tag{5.15}$$

This syntax means

$$A.R \supseteq \{s_1 \cup ... \cup s_k | \left(s_i \in members\left(B_i.R_i\right) \wedge s_i \cap s_j = \varnothing\right), 1 \leq i \neq j \leq k\} \tag{5.16}$$

Consider the following example.

Example 5.2 *Company 1 ($C1$) defines a person to be on a committee if one manager authorizes it and two different supervisors say so. This can be represented using the following credentials: If*

$$members\left(C1.Managers\right) \supseteq \{\{Miller\}, \{Paine\}\}$$
$$members\left(C1.Supervisors\right) \supseteq \{\{OldFather\}, \{Hamilton\}, \{Ferguson\}\}$$

and

$$C1.Temp1 \leftarrow C1.Supervisors \otimes C1.Supervisors$$
$$C1.Temp2 \leftarrow C1.Managers \odot C1.Temp1$$
$$C1.Committee \leftarrow C1.Temp2.Committee$$

Then we have

$$\begin{aligned} members\,(C1.Temp1) \supseteq\ & \{\{Miller, Hamilton\}, \{Miller, Ferguson\}, \\ & \{Hamilton, Ferguson\}\} \\ members\,(C1.Temp2) \supseteq\ & \{\{Miller, Hamilton\}, \{Miller, Ferguson\}, \\ & \{Miller, Hamilton, Ferguson\}, \\ & \{Miller, Hamilton, Paine\}, \\ & \{Miller, Ferguson, Paine\}, \\ & \{Hamilton, Ferguson, Paine\}\} \end{aligned}$$

And if

$$\begin{aligned} Miller.Committee &\leftarrow Miller \\ Miller.Committee &\leftarrow Hamilton \\ Hamilton.Committee &\leftarrow Hamilton \\ Hamilton.Committee &\leftarrow Ferguson \\ Hamilton.Committee &\leftarrow Paine \\ Ferguson.Committee &\leftarrow Ferguson \\ Ferguson.Committee &\leftarrow Paine \\ Paine.Committee &\leftarrow Paine \end{aligned}$$

Then one can conclude that

$$members\,(C1.Committee) \supseteq \{Hamilton, Paine\}$$

but one cannot conclude

$$members\,(C1.Committee) \supseteq \{Miller\}$$

or

$$members\,(C1.Committee) \supseteq \{Ferguson\}$$

5.1.1 RT^D Role Activations

Role activations allow authorization delegation from user-to-session and process-to-process. There must be an original principal-to-session delegation. These delegations

are different in that an authorization is being delegated for the lifetime of the session or process and the rights of a role are being passed to a session or process instead of a principal being put in a role. This is like a power of attorney contract for a limited amount of time or for performing a specific duty.

Delegation Credential: This rule means that B_2 is acting for D as $A.R$ if B_1 is acting for D as $A.R$.

$$B_1 \xrightarrow{D \text{ as } A.R} B_2 \tag{5.17}$$

The process of *acting for* ends (or starts) with

$$D \xrightarrow{D \text{ as } A.R} B_1 \tag{5.18}$$

where we have B_1 is acting for D as $A.R$ if D is in the role $A.R$.

Another form is when you delegate every activation in which you are acting for D

$$B_1 \xrightarrow{D \text{ as } all} B_2 \tag{5.19}$$

This rule means that if B_1 is acting for D it is activating those roles, thus B_2 is also acting for D in any role.

The last form activates all.

$$B_1 \xrightarrow{all} B_2 \tag{5.20}$$

This means that B_1 is activating every delegation. So if B_1 is acting, so is B_2.

Besides delegating role activations, RT^D also has a syntax for requesting a role activation.

$$B \xrightarrow{activation} req \tag{5.21}$$

where $activation$ is any one of the activations above and req is a dummy variable having a $RequestID$.

5.2 Translations

The *Trust Management* language allows other types besides integers and uses these types only in safe ways.

- Integers
- Closed enumeration types where the members are statically declared.
- Open enumeration types where the members are not statically declared.
- Floats which are really arbitrary precision rational numbers.

Because *Constraint Datalog* over addition constraints is a complete language it is enough to translate the delegations above into *Constraint Datalog*. It is possible to translate everything into integers and write every delegation and activation in terms of integers. In fact every translation is straight forward with the exception of manifold roles.

To handle manifold roles we define the principal being assigned in the translations below to be a $setid$, and an additional relation $set\,(setid, principal)$ which maps $setid$'s to the principals or other values contained in the set. Thus we translate relations containing sets to nested relations. This translation works well for all but the role product operators as we will see.

5.2.1 Simple member

From : $A.r\,(h_1, ..., h_n) \leftarrow D$
To : $\overline{r}\,(A, D, x_1, ..., x_k) :- \psi$

5.2.2 Simple containment

From : $A.r\left(h_1, ..., h_n\right) \leftarrow B.r_1\left(s_1, ..., s_m\right)$
To : $\overline{r}\left(A, y, x_1, ..., x_k\right) : -\ \overline{r_1}\left(B, y, x_{1,1}, ..., x_{1,k_1}\right),\ \psi$

5.2.3 Linking containment

From : $A.r\left(h_1, ..., h_n\right) \leftarrow A.r_1\left(s_{1,1}, ..., s_{1,m_1}\right).r_2\left(s_{2,1}, ..., s_{2,m_2}\right)$
To : $\overline{r}\left(A, y, x_1, ..., x_k\right) : -\ \overline{r_1}\left(A, z, x_{1,1}, ..., x_{1,k_1}\right), set(z, z'), \overline{r_2}\left(z', y, x_{2,1}, ..., x_{2,k_2}\right),\ \psi$

5.2.4 Intersection containment

From : $A.r\left(h_1, ..., h_n\right) \leftarrow A_1.r_1\left(s_{1,1}, ..., s_{1,m_1}\right) \cap ... \cap A_l.r_l\left(s_{l,1}, ..., s_{l,m_l}\right)$
To : $\overline{r}\left(A, y, x_1, ..., x_k\right) : -\ \overline{r_1}\left(A_1, y, x_{1,1}, ..., x_{1,k_1}\right), ..., \overline{r_l}\left(A_l, y, x_{l,1}, ..., x_{l,k_l}\right),\ \psi$

5.2.5 Simple delegation

From : $A.r\left(h_1, ..., h_n\right) \Leftarrow B : C.r_2\left(s_1, ..., s_m\right)$
To : $\overline{r}\left(A, y, x_1, ..., x_k\right) : -\ \overline{r_1}\left(B, y, x_{1,1}, ..., x_{1,k_1}\right), \overline{r_2}\left(C, y, x_{2,1}, ..., x_{2,k_2}\right),\ \psi$

5.2.6 Linking delegation

From : $A.r \Leftarrow A.r_1 : C.r_2$
To : $r\left(A, y, x_1, ..., x_k\right) : -\overline{r_1}\left(B, y, x_{1,1}, ..., x_{1,k_1}\right), \overline{r_2}\left(C, y, x_{2,1}, ..., x_{2,k_2}\right),\ \psi$

Example 5.3 *Consider the scenario in Figure 5.1. SciOrg gets funding from NSF so it allows NSF to have simple delegation of the SciOrgDB role to NSF. NSF allows research from UNL to access data if they are also a member of the NSF_Fellow role. Nebraska Department of Agriculture researchers also do research at UNL and are members of the researcher role.*

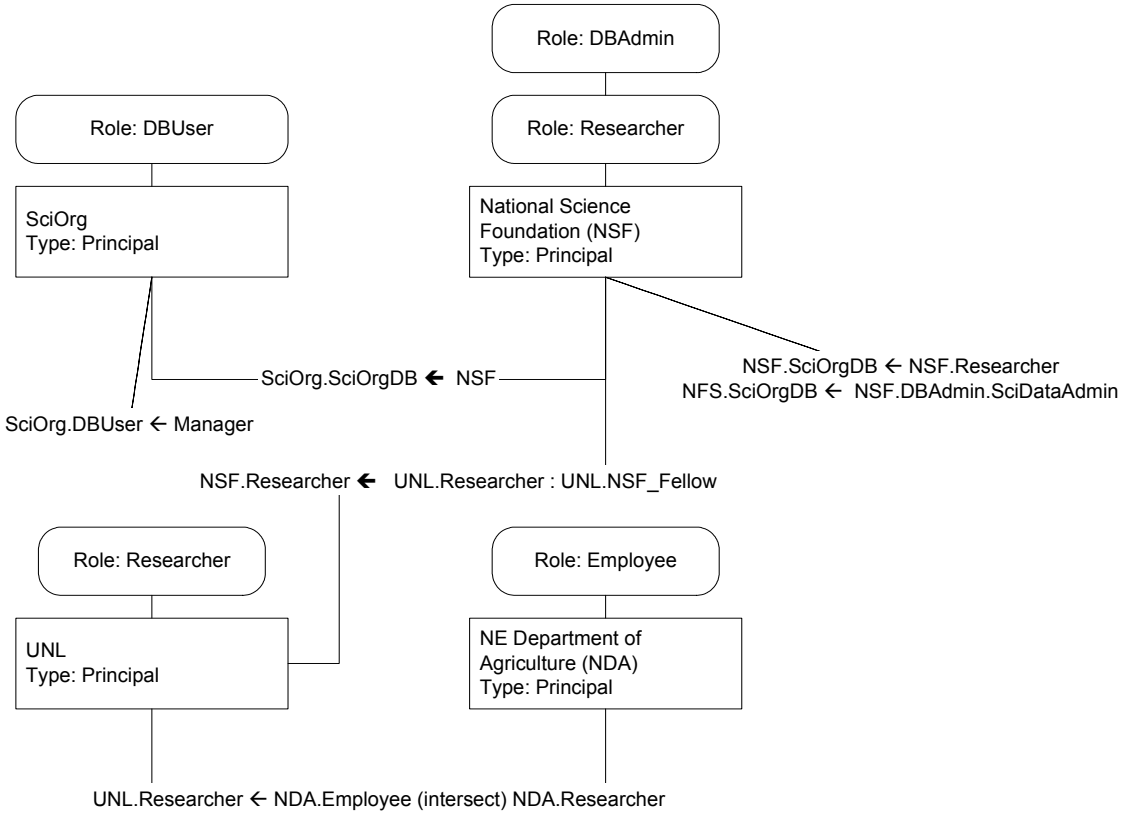

Figure 5.1: Role-based Trust Management Example

The statements in Figure 5.1 can be extended to include parameters as shown in Figure 5.2.

We categorize these rules in Figure 5.2 respectively as *Simple Member*, *Simple Containment*, *Linking Containment*, *Intersection Containment*, *Simple Delegation* and

$SciOrg.DBUser \leftarrow Manager(host = descendant(\text{“}sciorg.org\text{”}))$
$NSF.SciOrgDB \leftarrow NFS.Researcher$
$NSF.SciOrgDB \leftarrow NSF.DBAdmin.SciDataAdmin(time = [1700..2400])$
$UNL.Researcher \leftarrow NDA.Employee \cap NDA.Researcher$
$SciOrg.DBUser \Leftarrow NSF$
$NSF.Researcher \Leftarrow UNL.Reasearcher : UNL.NSF\_Fellow(level = \text{“}Ph.D\text{”})$

Figure 5.2: SciOrg RT_1^C rules

Linking Delegation. Using the rules above we translate these RT_1^C rules to *Constraint Datalog* in Figure 5.3.

$DBUser(C,D,host) :-$	$C = SciOrg,\ D = Manager,$
	$host << sciorg.org.$
$SciOrgDB(C,D,time) :-$	$C = \text{“}SciOrg\text{”}, E = \text{“}NSF\text{”}, SciOrgDB(E,D).$
$SciOrgDB(C,D,time) :-$	$C = \text{“}NFS\text{”}, E = \text{“}NFS\text{”}, Researcher(E,D).$
$SciOrgDB(C,D,time) :-$	$C = \text{“}NSF\text{”}, E = \text{“}NSF\text{”}, DBAdmin(E,F),$
	$SciDataAdmin(F,D), time >= 1700, time <= 2400.$
$Researcher(C,D,level) :-$	$C = \text{“}NSF\text{”}, E = \text{“}UNL\text{”}, Researcher(E,D),$
	$NSF\_Fellow(E,D), level = \text{“}Ph.D\text{”}.$
$Researcher(C,D,level) :-$	$C = \text{“}UNL\text{”}, E = \text{“}NDA\text{”}, Employee(E,D),$
	$Researcher(E,D).$

Figure 5.3: SciOrg *Constraint Datalog* Rules

5.2.1 O-set delegation

These translations are the same as the translations of the delegations above, except that the roles become *o-sets*.

5.2.2 Manifold roles

RT^T modifies the notion of elements of a role from being principals to sets of principals.

We introduce a predicate $set\,(id, value)$ that uniquely identifies the values in any set. This does not change the representation of delegation in Datalog for any of the above rules. However, it is necessary for us to define the role product operators $\odot$ and $\otimes$.

$$\begin{array}{lll} \text{From :} & A.R \leftarrow B_1.R_1 \odot ... \odot B_k.R_k & \\ \text{To:} & isMember(z, R, A) :- & role(A, R), \\ & & isMember(z_1, R_1, B_1), role(B_1, R_1), \\ & & ... \\ & & isMember(z_k, R_k, B_k), role(B_1, R_1), \\ & & set_k\,(z, z_1, ..., z_k) \end{array}$$

Where set_k is a new predicate symbol defined as follows.

Definition 5.4 [14] *set_k takes $k+1$ entity collections as arguments and $set_k\,(s, s_1, ..., s_k)$ is true if and only if $s = s_1 \cup ... \cup s_k$. When s_i is an entity it is treated as a single-element set*

$$\begin{array}{lll} \text{From :} & A.R \leftarrow B_1.R_1 \otimes ... \otimes B_k.R_k & \\ \text{To:} & isMember(z, R, A) & role(A, R), \\ & & isMember(z_1, R_1, B_1), role(B_1, R_1) \\ & & ... \\ & & isMember(z_k, R_k, B_k), role(B_k, R_k) \\ & & niset_k\,(z, z_1, ..., z_k) \end{array}$$

Where $niset_k$ is a new predicate symbol defined as follows.

Definition 5.5 [14] *$niset_k$ takes $k+1$ entity collections as arguments and $niset_k\,(s, s_1, ..., s_k)$ is true if and only if $s = s_1 \cup ... \cup s_k$ and for any $1 \leq i, j \leq k$ and $i \neq j, s_i \cap s_j = \varnothing$. When s_i is an entity it is treated as a single-element set*

We could easily allow enumeration types and define these predicate symbols. It is

```
R1(A,s) :- A=1, s=2.                    uniqueID(x,y,z) :- x=2,y=3,z=1.
R1(A,s) :- A=1, s=5.                    uniqueID(x,y,z) :- x=2,y=4,z=2.
R2(A,s) :- A=1, s=2.                    uniqueID(x,y,z) :- x=2,y=5,z=3.
R2(A,s) :- A=1, s=3.                    uniqueID(x,y,z) :- x=3,y=4,z=4.
R2(A,s) :- A=1, s=4.                    uniqueID(x,y,z) :- x=3,y=5,z=5.
R(id,A,s) :- R1(A,s), id=1.             uniqueID(x,y,z) :- x=4,y=5,z=6.
R(id,A,s) :- R2(A,s), id=2.             uniqueID(x,y,z) :- x=3,y=2,z=1.
set(i,v) :- i=2, v=2.                   uniqueID(x,y,z) :- x=4,y=2,z=2.
set(i,v) :- i=3, v=3.                   uniqueID(x,y,z) :- x=5,y=2,z=3.
set(i,v) :- i=4, v=4.                   uniqueID(x,y,z) :- x=4,y=3,z=4.
set(i,v) :- i=5, v=5.                   uniqueID(x,y,z) :- x=5,y=3,z=5.
UnionListA(x) :- x=1.                   uniqueID(x,y,z) :- x=5,y=4,z=6.
UnionListA(x) :- x=2.                   uniqueID(x,y,z) :- x=2,y=2,z=7.

setA(id,e) :- UnionListA(x), UnionListA(y), x-y<0, R(x,A,s1), R(y,A,s2),
              set(s1,e), uniqueID(s1,s2,id).
setA(id,e) :- UnionListA(x), UnionListA(y), y-x>0, R(x,A,s1), R(y,A,s2),
              set(s2,e), uniqueID(s2,s1,id).
```

Table 5.1: Role Dot Product

possible to define the product operators using only integer constraints. Consider the following example for the $\odot$ product.

Example 5.6 *Consider $R1$ and $R2$ to be manifold roles containing singleton set elements as follows:*

$$R1 = \{\{2\}, \{5\}\}$$
$$R2 = \{\{2\}, \{3\}, \{4\}\}$$

Notice that the set names coincide with the element, but that does not need to be the case. The Constraint Datalog program in Table 5.1 *calculates $R1 \odot R2$. The Unique ID's are required because we are creating new set elements to include in a role. The output shown in Table* 5.2 *gives the results of the program*.

This program requires 2^n tuples for unique identifiers. It also requires 2^{k-1} different rules like the last two rules in Table 5.1, because we must account for the different

setA(id, e) :- id = 1, e = 2.
setA(id, e) :- id = 1, e = 3.
setA(id, e) :- id = 2, e = 2.
setA(id, e) :- id = 2, e = 4.
setA(id, e) :- id = 3, e = 5.
setA(id, e) :- id = 3, e = 2.
setA(id, e) :- id = 5, e = 5.
setA(id, e) :- id = 5, e = 3.
setA(id, e) :- id = 6, e = 5.
setA(id, e) :- id = 6, e = 4.
setA(id, e) :- id = 7, e = 2.

Table 5.2: Role Dot Product Results

orderings possible. It may be possible to lower this last restriction with some kind of ordering.

5.2.1 Role Activations

This is a single activation by B_1 that says B_2 is acting for D as $A.R$ where R is a specified role.

From: $B_1 \xrightarrow{D \text{ as } A.R} B_2$
To: $forRole\left(B_2, D, A, R\right) :- forRole\left(B_1, D, R\right),\ role\left(A, R\right).$

This role encompassing activation by B_1 says that B_2 is acting for D in any role that B_1 is acting for where r is a role variable.

From: $B_1 \xrightarrow{all \text{ as } A.R} B_2$
To: $forRole\left(B_2, D, A, r\right) :- forRole\left(B_1, D, r\right),\ role\left(A, r\right).$

This all encompassing activation by B_1 says that if B_1 is acting for any entity in any role, B_2 also is activated for these roles. Here a and r are both variables.

From: $B_1 \xrightarrow{all} B_2$
To: $forRole\left(B_2, D, a, r\right) :- forRole\left(B_1, a, r\right),\ role\left(a, r\right).$

A request for an activation is similar.

From: $B_1 \xrightarrow{D \text{ as } all} req$
To: $forRole\left(ReqID, D, a, r\right) :- forRole\left(B_1, D, a, r\right),\ role\left(a, r\right).$

Statement Type	RT Language	Constraint Datalog	Explanation
simple member	$2+n$	$3+k$	$n \approx k$
simple containment	$2+n+m$	$3+k_1{+}k_2$	$n+m \approx k_1{+}k_2$
linking containment	$3+n+m_1{+}m_2$	$7+k+k_1{+}k_2$	$n+m_1{+}m_2$ $\approx k+k_1{+}k_2$
intersection containment	$l+1{+}\sum_1^l s_{l,m_l}$	$2(l+1)+\sum_1^l x_{l,k_l}$	$\sum_1^l s_{l,m_l} \approx \sum_1^l x_{l,k_l}$
simple delegation	$3+n+m_1{+}m_2$	$7+k+k_1{+}k_2$	$n+m_1{+}m_2$ $\approx k+k_1{+}k_2$
linking delegation	$3+n+m_1{+}m_2$	$7+k+k_1{+}k_2$	$n+m_1{+}m_2$ $\approx k+k_1{+}k_2$
o-set delegation	same as above	same as above	same as above
manifold roles	$4(k+1)$	$\sum_i \lvert n_i \rvert +4(k+1)$	n_i is the number of enumeration elements
activations	k	$k+1$	nearly identical

Table 5.3: Number of Variables Required

The notation for role activations differentiates it from delegations. There is no difference in the Datalog rules that they are translated to, except the relation name *forRole* and the dynamic nature of activations.

5.3 Comparing the Complexity

Here we are comparing the complexity of the RT family of languages with the rules as they are translated into *Constraint Datalog*. In analyzing this we use two different metrics to compare the language to the semantics. The first metric compares the number of variables required in *Constraint Datalog* to the number of variables in RT. The second metric compares the number of rules required in *Constraint Datalog* to the number of rules in RT. Another way to describe this is to say that we compare the size of the program needed to execute the statements using the logic rules found in [14] with the different types available and using *Constraint Datalog* over integer addition constraints without additional types.

In Table 5.3 all the statements have similar numbers of variables except the manifold roles. This occurs because in the role product operators $\odot$ and $\otimes$, we have to manipulate the actual elements in the set instead of just the set IDs. Since we do not have enumeration (set) elements, the number of variables in this case is much greater for *Constraint Datalog*. Since the rules are designed with logic in mind, it is not surprising that there is little difference between the number of variables in the rule and the number of variables needed to implement the rule in *Constraint Datalog*.

When comparing the number of *Constraint Datalog* rules needed for each RT rule, we again see a 1-to-1 correspondence except in manifold role product operations. Because we are implementing sets of sets using Datalog with addition constraints only, we have to build the operators without using set notation. Although the program in Table 5.1 runs, it clearly could be written in one rule if MLPQ [19] had enumeration types.

We conclude that the policies in *RT* have a coinciding number of variables and rules except for separation-of-duty policies.

Chapter 6: Expressiveness using Addition Constraints in Security

It does not take much imagination to find several security situations that may use addition constraints or some type of Diophantine equation constraint to authorize some request. In general, if we have one or more rating organizations that assign an integer level of confidence to an individual, then it may be useful for a different organization to calculate an acceptable confidence level based on other organizations' rating. It may be especially useful if the rating system in use is scaled differently than the rating system of a reference organization.

As another example, we note that *Trust Management* can extend beyond security to include trust as it relates to E-mail from individuals. We might also extend it to gathering and analyzing different ratings form E-mail servers.

Consider the example of a threshold policy. Suppose an FBI employee also has an NSA clearance. The CIA may wish to provide individuals with both FBI and NSA clearances a level of security based on the following credential

$$\begin{aligned} CIA\left(id, l_l\right) :- \quad & FBI\left(id, l_1\right),\ NSA\left(id, l_2\right), \\ & l_l \leq \tfrac{2l_1+l_2}{3} < l_u,\ l_u - l_l = 1, \\ & VisitLevel\left(l_l, l_u\right),\ l_l < 7. \end{aligned}$$

Clearly we can write the first constraint as

$$3l_l \leq 2l_1 + l_2 < 3l_u$$

which is expressed as an addition constraint in *Constraint Datalog*. Using the difference and multiplication rules from Table 4.1, we get the following valid rule:

$$\begin{aligned} CIA\left(ID, l_l\right) :- \quad & FBI\left(id, l_1\right), NSA\left(id, l_2\right), \\ & 3l_l - 2l_1 + l_2 \leq 0, \\ & 2l_1 + l_2 - 3l_u < 0, \\ & l_u - l_l = 1, \\ & VisitLevel\left(l_l, l_u\right). \end{aligned}$$

Threshold policies are one of the types of policies that other *Trust Management* languages have difficulty expressing. The concept of using an approximately complete, safe language is very important in that respect. Now it is possible to express any type of policy including all types of threshold polices if we base a *Trust Management* language on *Datalog* with addition constraints and approximation.

Chapter 7: Conclusions

We reviewed the concepts of *Trust Management*, gave a brief overview of several *Trust Management* systems and described the environment and its requirements. We proposed a *Turing-complete* language as a solution to the problem of continually adding features to existing languages. The halting problem associated with complete languages prevents this possibility unless a we can overcome it. Approximation in *Constraint Datalog* over addition constraints solves this problem. The theory was given and a detailed fractal example demonstrated the approximation theory. We then showed that Diophantine equations are expressively equivalent to Turing Machines and cited Matiyasevich's work in proving this. Based on approximation and Diophantine equations being *Turing-complete*, we proposed that *Constraint Datalog* over addition constraints be used as an approximately complete, safe language. This was shown by demonstrating that any Diophantine equation can be expressed by using *Constraint Datalog* over addition constraints. We gave a detailed look at translating the Role-based *Trust Management* family of languages into *Constraint Datalog* and showed that the rules in RT have similar complexity to the rules in *Constraint Datalog*. Finally, additional reasons for using *Constraint Datalog* with addition constraints was given along with an example of a threshold policy.

Finally, we mention some directions for future work. First, adding the ability to update a tuple inside the body of a rule would simplify many operations, but would need a theoretical basis in safety. Second, different kinds of policy constructs, such as mutual exclusion of roles, need to have semantics defined in *Constraint Datalog*.

Bibliography

[1] BLAZE, M., FEIGENBAUM, J., AND LACY, J. Decentralized trust management. Tech. Rep. 96-17, AT and T Research, 1996.

[2] CHU, Y.-H., FEIGENBAUM, J., LAMACCHIA, B., RESNICK, P., AND STRAUSS, M. REFEREE: Trust management for Web applications. *Computer Networks and ISDN Systems 29*, 8–13 (1997), 953–964.

[3] DANTSIN, E., EITER, T., GOTTLOB, G., AND VORONKOV, A. Complexity and expressive power of logic programming. In *IEEE Conference on Computational Complexity* (1997), pp. 82–101.

[4] DIFFIE, W., AND HELLMAN, M. E. New directions in cryptography. *IEEE Transactions on Information Theory IT-22*, 6 (1976), 644–654.

[5] ELLISON, C. M., FRANTZ, B., LAMPSON, B., RIVEST, R., THOMAS, B., AND YLONEN, T. SPKI certificate theory. RFC 2693, Internet Engineering Task Force, Sept. 1999. See `http://www.ietf.org/rfc/rfc2693.txt`.

[6] FREUDENTHAL, E., PESIN, T., PORT, L., KEENAN, E., AND KARAMCHETI, V. drbac: Distributed role-based access control for dynamic coalition environments, 2002.

[7] GLEICK, J. *Chaos, Making a New Science*. Viking Penguin Inc., 1987.

[8] GRANDISON, T., AND SLOMAN, M. A survey of trust in internet application. *IEEE Communications Surveys and Tutorials 3*, Fourth Quarter (2000).

[9] HERZBERG, MASS, MIHAELI, NAOR, AND RAVID. Access control meets public key infrastructure, or: Assigning roles to strangers. In *RSP: 21th IEEE Computer Society Symposium on Research in Security and Privacy* (2000).

[10] HOPCROFT, J. E., AND ULLMAN, J. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Publishing Co., Reading Massachusetts, 1979.

[11] IBM. IBM trust establishment - policy language, October 2003. See http://www.haifa.il.ibm.com/projects/software/e-Business/TrustManagement/PolicyLanguage.html.

[12] LI, N. *Delegation Logic: A Logic-based Approach to Distributed Authorization*. PhD thesis, New York University, September 2000.

[13] LI, N., AND MITCHELL, J. Understanding SPKI/SDSI using first-order logic. In *To Appear in IEEE Computer Security Foundations Workshop* (2003).

[14] LI, N., MITCHELL, J. C., AND WINSBOROUGH, W. H. Design of a role-based trust management framework. In *Proc. IEEE Symposium on Security and Privacy, Oakland* (May 2002).

[15] MATIYASEVICH, Y. V. *Hilbert's Tenth Problem*. MIT Press, 1993.

[16] PENROSE, R. *The Emperor's New Mind*. Oxford University Press, 1989.

[17] REVESZ, P. *Introduction to Constraint Databases*. Springer-Verlag, 2002.

[18] REVESZ, P. A retrospective on constraint databases. In *Proceedings of the Paris C. Kanellakis memorial workshop on Principles of computing and knowledge* (2003), ACM Press, pp. 12–27.

[19] REVESZ, P., CHEN, R., KANJAMALA, P., LI, Y., LIU, Y., AND WANG, Y. The MLPQ/GIS constraint database system. In *ACM SIGMOD International Conference on Management of Data* (2000).

[20] RIVEST, R. L., SHAMIR, A., AND ADELMAN, L. M. A METHOD FOR OBTAINING DIGITAL SIGNATURES AND PUBLIC-KEY CRYPTOSYSTEMS. Tech. Rep. MIT/LCS/TM-82, National Science Foundation, 1977.

[21] SANDHU, R. S., COYNE, E. J., FEINSTEIN, H. L., AND YOUMAN, C. E. Role-based access control models. *IEEE Computer 29*, 2 (1996), 38–47.